Optimizing Tensegrity Gaits Using Bayesian Optimization

By

James M. Boggs

* * * * * * * * *

Submitted in partial fulfillment

of the requirements for

Honors in the Department of Computer Science

UNION COLLEGE

June, 2018

Abstract

BOGGS, JAMES M. Optimizing Tensegrity Gaits Using Bayesian Optimization. Department of Computer Science, June, 2018.

ADVISOR: Prof. John Rieffel

We design and implement a new, modular, more complex tensegrity robot featuring data collection and wireless communication and operation as well as necessary accompanying research infrastructure. We then utilize this new tensegrity to assess previous research on using Bayesian optimization to generate effective forward gaits for tensegrity robots. Ultimately, we affirm the conclusions of previous researchers, demonstrating that Bayesian optimization is statistically significantly ($p < 0.05$) more effective at discovering useful gaits than random search. We also identify several flaws in our new system and identify means of addressing them, paving the way for more effective future research.

Contents

List of Figures

List of Tables

1 Introduction

The bulk of robots in use today are variations of so-called ‘hard robots’, robots which are made from inflexible materials like plastic or metal and are essentially rigid in nature. Although rigidity in robots can have many advantages, it also introduces certain weaknesses. When placed under compression, the bodies of hard robots maintain their form until they break catastrophically, and if they encounter a space which their bodies don’t fit into they are unable to navigate it. Soft robots address these problems by being morphologically flexible through various means. Tensegrity robots are an example of this, composed of rigid struts connected to each other by tensile links (usually springs), such that the opposing tensile forces of the links result in the structure taking on and maintaining a certain form (see Figure 1).

When unperturbed the tensegrity maintains this resting form, but the tensile nature of the links between the struts means that when a force is applied to the robot, it deforms and vibrates before eventually returning to its resting form. The ability of tensegrity robots to deform and reform gives them several advantages over traditional, rigid robots, including 1) greater resilience against mechanical shocks like drops or sudden impacts, 2) the ability to navigate through spaces which are inaccessible to its resting morphology, and 3) greater portability on account of significantly less mass and the ability to be compressed in transit.

These advantages make tensegrities particularly attractive options for planetary exploration [1]. Traditional rovers are costly to launch on account of their bulk, taking up a great deal of space in a launch vehicle and requiring more fuel to launch. Tensegrities are able to offer substantial savings on these costs: their volume is mostly air so they weigh very little, and they’re able to be flat-packed by de-stressing the springs. Additionally, their resilience to hard impacts means that less needs to be done to soften their landing on the planet, whereas the landing of a rover must be exceedingly gentle.

Although potentially quite useful, tensegrities face a challenge in terms of control systems. First, traditional robots are engineered to have very stable and predictable dynamics, both in terms of structure and their effectors. Rigid bodies and predictable effectors mean that human engineers can hand-design control systems using forward and inverse kinematics. Tensegrities, on the other hand, are highly dynamically complex and vibrationally active, making it essentially impossible to even simulate them accurately, let alone design reliable control structures. Second, robots often require many complex sensors and effectors to be useful in the real world, and in traditional robots the coordination between the many sensors and effectors is done by a central controller. However, the morphology of tensegrities tends to be poorly suited for a centralized on-board controller and the energy consumption due to such a controller would require large batteries which would counteract some of the advantages given by tensegrities.

1.1 Background

A significant amount of work has been done in the past decade or so to address these problems with the control of tensegrity structures. First, it is worth noting that researchers have developed two different methods for actuating the tensegrity, independent of gait generation. These two methods are adjusting the resting lengths of the tensile elements between struts or inducing vibration in the tensegrity. The former approach is by far the most widespread, and was the method used in [1, 2, 4, 3, 6, 11, 12, 16, 17, 20, 22, 21, 24]. Vibration-based locomotion is much more rare, seen only in research overseen Rieffel: [23, 14, 13].

1.1.1 Tensegrity Actuation

Paul et al. [20] were the first to propose the concept of tensegrity-based robots and utilized cable length actuation to produce movement in their robot. They developed two simple tensegrity robots, one with three bars and one with four, in simulation using the Open Dynamics Engine, simulating the struts as rigid cylinders and the springs as collinear forces at the end of each strut the spring connected. The movement of the tensegrity was accomplished by repeatedly adjusting the resting lengths of the springs in a loop. This proved successful, and their simulated robots moved effectively within the virtual world. This translated successfully to the physical world, where an they got an embodied tensegrity robot to move effectively with the same technique, thereby showing that tensegrity robots are plausible. Iscen et al. [12] is another notable example of the success of cable length actuation as a locomotive method. In their paper, Iscen et al. demonstrate a particular kind of gait they call “flop-and-roll” which works by lengthening cables in a large six-bar tensegrity so that the robot flops forward due to gravity, then tightening cables so the robot rolls itself forward into a new position.

The use of vibration to actuate a tensegrity has so far been limited, used only by Khazanov et al. [14, 13], Rieffel and Mouret [23], and Böhm and Zimmermann [5]. However, their papers demonstrate that vibration is a plausible method of locomotion for tensegrity robots in both simulation and the physical world. In Khazanov et al. [14, 13] and Rieffel and Mouret [23], a small, embodied, six-bar tensegrity was driven by three small DC motors with off-centered weights. These motors induced regular oscillation in the struts to which they are attached which in turn vibrated the tensegrity and caused it to move. They show that different combinations of motor speeds induced oscillations with different characteristics, and that this in turn changed the nature of the tensegrity’s motion. Thus, the actuation of the tensegrity can be adjusted by adjusting motor speeds. Overall, their experiments demonstrated that the use of vibration produces much faster locomotion than cable length actuation is capable of inducing, and that gaits based on vibration can still be generated effectively. Böhm and Zimmermann [5] took a significantly different approach and worked with a tensegrity with angled struts rather than classic straight ones. The two angled struts were connected such that a pair of electromagnets at their inflection points could attract or repulse each other, and by varying this vibrations in the tensegrity could be produced.

1.1.2 Gait Generation

However, the primary focus of tensegrity research has not been on actuation but gait generation. The importance of studying gait generation in tensegrities stems from the near-impossibility of hand-designing effective gaits for them given their highly dynamic nature. Given this difficulty, researchers have needed to develop methods for generating effective gaits quickly and accurately, and have landed upon two primary methods. One method, explored by Bliss [2], Bliss et al. [4, 3], Caluwaerts et al. [6], Mirletz et al. [16, 17], Rieffel et al. [21], and Tietz et al. [24], is to train neural networks to control the actuation of a tensegrity. Agogino et al. [1], Iscen et al. [11, 12], Khazanov et al. [13], and Paul et al. [20] investigate the other option: using evolutionary algorithms to produce effective gaits. A new approach to gait-generation which also uses an optimization algorithm is demonstrated by Rieffel and Mouret [23], who use Bayesian optimization to identify effective gaits. This method offers a potentially faster way to discover effective gaits that either evolutionary algorithms or neural networks, and the focus of this paper will be investigating the efficacy of Bayesian optimization for this purpose.

Neural networks have, in the last decade and a half, become one of the most popular machine learning techniques, and it is unsurprising that they have been explored as potential means of generating gaits for tensegrity robots. In particular, small neural networks have been used in biologically-inspired setups to produce central pattern generators – cyclical neural networks which output a regular pattern. The first work combining neural networks and tensegrity robots was done by Rieffel et al. [21]. Their idea was to have each strut be an independent spiking neural network, a kind of artificial neural network which is time-sensitive. Nodes in a spiking neural network do not use sigmoid functions to determine their output. Instead, they sum weighted inputs with a persistent counter, and if the value of the counter is over a given threshold the neuron "spikes." If the counter is not over the threshold, the counter is decremented by a fixed decay value. The summing and decaying over time means SNNs are able to adjust their firing rate based on input, which makes them ideal for motor actuation. Each strut had its own small SNN which had two inputs nodes sensing the tension of a spring at each end, two hidden nodes, and two output nodes. The firing of the output nodes dictated when a cable was contracted, with each node controlling the cable length of a different attached cable. Rieffel et al. demonstrated that such networks could be trained to work together across the tensegrity and produce effective forward gaits, at least in simulation. Although Rieffel et al. never mention it by name, the core concept was similar to the central pattern generators explicitly embraced in future works.

Starting with Bliss [2], the use of neural networks was explicitly coupled with the use of central pattern generators in research on tensegrities. The concept of the central pattern generator comes from biology, where CPGs are often found. In nature CPGs are systems of neurons which produce regular, rhythmic pulses and are often associated with movement or motor actions. Bliss notes that "neural circuit responsible for the production of rhythmic signals controlling an animal's wings, legs, arms, or tail in the act of locomotion," and that " CPGs are responsible for more than simply locomotion, including control of chewing, scratching, breathing, and heart beating." [2, p. 11]. In relation to tensegrity control systems, CPGs are often implemented as small neural networks, sometimes as small as two neurons, which oscillate firing. The pattern generated by the CPG is used to drive the locomotion of the tensegrity, either by adjusting cable lengths directly as in [2, 4, 3, 6] or by altering the parameters of an impedance controller as in [16, 17, 24].

Thomas Bliss and his team [2, 4, 3] use a simple, two-neuron CPG called a reciprocal inhibition oscillator (RIO) to actuate cable lengths in a tensegrity. The tensegrity in question is both simple and significantly different than standard tensegrities, being designed to be flexible only on two axes rather than three. The tensegrity used is embodied and supports a fin which is placed in a body of water, and the tensegrity is actuated with the goal of swimming forward by moving the fin. Two cables attached the the base of the tensegrity allow a controller to pull on one side or the other of the tensegrity, causing it to flex back and forth in a sort of flapping motion. The CPG determines when and to what extent each of the two cables is contracted, thus dictating the swimming motion of the tensegrity and fin. Bliss et al. use several different techniques to entrain a specific pattern in the CPG which was hand-designed prior to testing. In doing so, they demonstrate that CPGs can successfully be used to control the actuation of tensegrity cables, but don't directly address discovering effective CPGs.

The approach taken by Mirletz et al. [16, 17] and Tietz et al. [24] for their Tetraspine tensegrity was slightly different because the CPG controlled parameters of a local impedance controller on each strut, rather than directly actuating the robot. Their Tetraspine tensegrity was a non-standard class III tensegrity meant to abstractly model a real spine, and was composed of a handful of tetrahedral vertebrae connected to each other by tensile elements. It used cable length adjustment to move, with each cable's length controlled by a local impedance controller. This controller adjusted the length of its cable based on a length offset, a tension factor, and a velocity factor, and each impedance controller was wholly independent from the others. A series of interconnected CPGs were used to dictate the desired resting length of each cable, which in turn affected the operation of the impedance controller by altering the tension factor. Using this method, Mireletz et al. and Tietz et al. gave a simulated Tetraspine the ability to not only move forward but to successfully overcome uneven terrain and obstacles in its path.

Like Bliss et al., Caluwaerts et al. [6] used a CPG to control the actuation of individual springs. The CPG used by Caluwaerts et al. was larger than that used by Bliss et al., and the generator itself was of a different type, with Caluwaerts et al. using a Matsuoka oscillator rather than an RIO. A Matsuoka oscillator features sparsely connected integrating neurons with time fatigue as the means of generating its pattern. Rather than simulating numerous neurons, however, Caluwaerts et al. used a central function which generated a two-dimensional matrix where each column was a list of the spring length of a spring for the past k time steps. This matrix was then multiplied by a a weighting matrix to produce a final output matrix which assigned a new spring length to each actuated spring. Gaits were generated by optimizing the weighting matrix so that the constant feedback loop generated a particular pattern. Ultimately, Caluwaerts et al. successfully used the CMA-ES function to optimize the weight matrix to produce a forward movement in a six-bar tensegrity. Although this does demonstrate the efficacy of CPGs in actuating tensegrities, its larger contribution is demonstrating that morphological computation is quite useful.

Although using neural networks to enact central pattern generators is a promising line of research, it shares many of the traditional pitfalls neural networks face: slow training, large sample requirements, and expensive computations. More worryingly, on-board neural networks on tensegrities have not left the realm of simulation, and implementing such a neural network on an embodied tensegrity is a daunting task. In contrast, evolutionary algorithms, although slow, have been demonstrated to be effective in the real world. The evolutionary approach was the first taken in tensegrity research. As noted earlier, Paul et al. [20] adjusted cable lengths in a continuous cycle, with each cable actuator contracting once and being released once per cycle. The phase of this cycle, the duration the cable was contracted compared to released, the amplitude of the contraction, and the period of the cycle were encoded in a genetic algorithm whose fitness function was based on distance traveled by the robot from the origin. The evolution itself was performed and evaluated only in simulation, however. Although they did demonstrate the viability of tensegrities in an embodied robot, the gait used in that scenario was hand-coded.

Iscen et al. [11, 12] and Agogino et al. [1] used genetic algorithms to develop controllers for both simulated and physical cable-actuated tensegrities. The primary method of locomotion outline in their work is a rolling method they call ‘flop and roll,’ in which certain cables are actuated so that the tensegrity falls to a side, then others are actuated to bring it up-right with the fallen side as its new base. Initially, they evolved sine waves to control the length of the cables over time [11], but then seized on the flop and roll technique and evolved a controller designed to effect it [12] through local impedance controllers. These techniques proved successful, generating a rolling motion in any direction given. However, the method is slow and relies on damping, rather than embracing, the high dynamism of tensegrities.

Khazanov et al. [13] evolved gaits by adjusting the speeds of the weighted motors used to induce vibration in their tensegrity. They had three motors attached to three separate struts on their embodied tensegrity, and used a set of three numbers indicating the motor speed for each motor as the genotype to evolve. Just like [20, 11, 12, 1], Khazanov et al.'s experiments ultimately produced effective gaits. However, the speed of the robot relative to its body size was greater compared to other evolved gaits, likely due to the use of vibration rather than cable length adjustment as the means of actuation. Still, although successful, work on using evolutionary algorithms to generate effective gaits demonstrated the weaknesses of evolutionary algorithms in general: they require many trials, which can take a great deal of time.

A new development in the field of tensegrity gait generation has been the use of Bayesian optimization by Rieffel and Mouret [23]. Their research utilized a small, embodied tensegrity robot tethered to a desktop computer, which itself used an advanced motion-capture system to track the movements of the robot. Their research demonstrated that within only 30 iterations, Bayesian optimization with prior knowledge results in statistically-significantly better gaits than random search ($p < 0.0001$). In particular, their priors indicated that full motor speeds (in any direction) would lead to greater movement. In their work, they measured the distance traveled by the tensegrity over the course of three seconds, using only thirty iterations per run. Ultimately, they demonstrated a gait developed through Bayesian optimization which was capable of moving the tensegrity 1.15 body lengths per second.

1.2 Bayesian Optimization

Bayesian optimization is, as the name suggests, an optimization algorithm. That is, Bayesian optimization attempts to find the optimal input for some function, which is often the input which maximizes or minimizes the output value of the function. Bayesian optimization is specifically used for determining the optimal input to an unknown function which can be tested but not directly known. In other words, given some unknown function $f(x) : X \to y$ where X is the set of possible inputs and y is a measurable output, Bayesian optimization can find an input x which produces a close-to-optimal output y. Bayesian optimization does so by maintaining a probabilistic model of the function which gives, for any possible input, a probability distribution of possible outputs. Although there are a number of ways to represent the function model probabilistically, a commonly-used one and the one used in this research as well as the work of Rieffel and Mouret is representing the function as a Gaussian process. A Gaussian process models the function f by assigning a Gaussian distribution of possible values of $f(x)$ to every possible x. Since the distribution of possible values of $f(x)$ for any x is Gaussian, it is defined by a mean μ and a variance σ^2, both likely unique to x. This means the Gaussian process can be used to predict likely outcomes for any value x.

Bayesian optimization utilizes this as a means of estimating the shape of the unknown function it is optimizing. Starting with some prior knowledge, either given to the algorithm *a priori* or discovered through experimentation, Bayesian optimization generates a Gaussian process which fits the prior knowledge. Then Bayesian optimization refines its model by choosing an input based on its probability distribution using an acquisition function, which examines the Gaussian process and selects the most promising candidate based on criteria which vary between acquisition functions. It then gives this input to the function, measures the output, and fits its Gaussian process to this new datum. That is, it experiments based on probability-based hypotheses and refines its model based on the new empirical evidence it generates. Note that it is assumed that the unknown function is smooth, so that knowledge of the results of some inputs can be used to make judgments about the probability distributions of other inputs. As the algorithm generates more data points and refines its model, it has a better idea of which inputs are likely to be fruitful, and can thus hone in on an optimal solution.

Two acquisition functions are investigated in this research: 'expected improvement' and 'upper confidence bound (UCB).' The expected improvement acquisition function chooses the point which has the greatest chance of returning the highest payoff. Given τ, the best evaluation result yet seen and $\mu(x)$ and $\sigma(x)$, the mean and standard deviation of the Gaussian distribution at input x, the expected increase Δ of input x over the previous best input is given by the equations

$$z = \frac{\mu - \tau}{\sigma} \tag{1}$$

$$\Delta = (\mu - \tau) \cdot \text{cdf}(z) + \sigma \cdot \text{pdf}(z) \tag{2}$$

where cdf and pdf are the cumulative distribution function and probability density function on the normal distribution, respectively. In essence, the expected improvement function determines the difference between the expected mean and the best evaluation in terms of standard deviations from the mean, then uses this to calculate the actual difference between the expected value of $f(x)$ and τ likely to be seen.

Whereas the expected improvement function takes a very moderated approach focusing on the most likely outcome, the upper confidence bound function is more optimistic, and its level of optimism is tunable. The UCB function assumes with a variable level of confidence that for any input x, the value of $f(x)$ will be above average. More formally, given $\mu(x)$ and $\sigma(x)$ as well as a confidence parameter β, the upper confidence bound ucb_β of $f(x)$ is given by

$$ucb_\beta = \mu + \sigma \cdot \beta \tag{3}$$

This function simply multiplies the standard deviation of the Gaussian distribution over $f(x)$ by a confidence level and adds it to the mean expected value. The idea is that the value β describes how many standard deviations above the mean we expect each value to perform. The value of β can be tuned as a hyperparameter, with lower values focusing on exploitation while higher values focus on exploration.

1.3 Research Question

Although the work of Rieffel and Mouret [23] is promising, it is important to be able to replicate the results of their experiments. In addition to purely replicating their work, several aspects can be altered in order to provide a more robust demonstration of the efficacy of Bayesian optimization. First, using different hardware than their work will ensure that their results are not hardware dependent. This variation comes in two ways: the tracking system and the robot itself. The tracking system used by Rieffel and Mouret, an eight-camera OptiTrack Prime setup, is a highly-price and expensive motion capture system capable of millimeter-level tracking. This grants the optimization algorithm a level of precision which it may be unrealistic to expect in the real world, particularly if training must occur outside of a lab. Additionally, the tensegrity robot used in their work was a small, primitive one which was controlled and powered via tethers made of magnet wire which directly connected the robot to a computer, the only on-board electronics were the motors used for actuation. In reality, tensegrities are likely to have various electronic components used for sensory and control purposes. Additionally, any genuinely useful tensegrity would need to be wireless, and thus require on-board power as well as control. The extra electronics could affect the weight and dynamics of the tensegrity in unpredictable ways, and Bayesian optimization must be able to account for that.

Second, Rieffel and Mouret had each optimization run perform only 30 trials, each lasting only three seconds. Although the data they gathered was statistically significant, there may be more effective gaits which the optimization functions did not discover due to the restricted number of trials, and the full behavior of a gait might not have become apparent with only three-second experiments. As such, the question this research seeks to address is **to what extent, if any, are we able to replicate the results of Rieffel and Mouret using a low-cost setup, an untethered tensegrity, and extended optimization runs?**

2 System Design

In order to address the first concern given in Section 1.3, and to demonstrate the versatility, utility, and potential of tensegrity robots in general, we designed and manufactured a modular, wireless, and instrumented tensegrity robot over the course of approximately two years. Additionally, a new testing rig was designed and built and a tracking algorithm was designed for tracking the tensegrity.

2.1 Physical Robot Design

The final design features 18.5cm long struts laser-cut from transparent black acrylic, which are wider at the center in order to accommodate a micro-controller and other electronics mounted on a protoboard. These struts are capped on each side by metal disks with 8 regularly spaced holes, into which springs are fitted. Each of the springs in the tensegrity are identical steel springs with a resting length of 4cm, and the connections between struts made by the springs forms the pre-stress stable ball-like configuration seen in Figure 1, which is approximately 22.5cm across.

Each strut features two mounting holes which can be used to affix a strut controller. A strut controller comprises a protoboard onto which several electronic components are mounted which together provide means of controlling the motion of the strut. At the core of a strut controller is the RFD77201 Simblee Bluetooth-enabled micro-controller. The RFD77201 micro-controller is based on an Arduino processor and is capable of running programs written in the Arduino programming language. It features seven GPIO pins which can be used to receive sensor input or send instructions to effectors. On the strut controllers used in this research the RFD77201 is connected to a DFRobot SEN0142 six-axis accelerometer and gyroscope and a Pololu DRV8838 Single Brushed DC Motor Driver Carrier, which in turn controls a Pololu brushed DC motor with an off-center mass attached to the drive. Finally, an RFD22130 Micro-SD shield may optionally be attached to the micro-controller to allow data storage. Powering the strut are two 3.7v 150 mAh lithium-ion batteries attached to the underside of the strut by 3d-printed clips. The entire assemblage can be seen in Figure 2.

The RFD77201 micro-controller runs Arduino control code which gives it several basic functionalities. It's primary feature is the ability to interface with a 'supervisor' computer via Bluetooth in order to run trials. This process involves the micro-controller signaling to the supervisor that it is ready to run a trial, the supervisor replying, when ready, with a motor frequency to try, the micro-controller spinning its motor at the allotted speed for ten seconds, and then the micro-controller signaling to the supervisor that the trial is complete. In addition to the ability to remotely run tests, the control code also allows for reading data from an accelerometer via I^2C and writing data to a micro-SD card via SPI, and is able to signal to the supervisor on startup whether it can utilize either option.

The design of the strut is the result of collaborative work between a group of engineers and the author. Riley Konsella '17, a computer engineer, worked to design a laser-cutable strut shape which could be used as the base for a modular tensegrity strut while simultaneously designing the electronic component responsible for enabling wireless control. His work was continued by Mitchell Clifford ’18, an electrical engineer, and Kentaro Barhydt ’18, a mechanical engineer, who together designed the final version of the strut and on-board electronics described above. The author worked concurrently with all three engineers to develop the control code for the RFD77201 micro-controller described above over the course of the two years.

The resulting tensegrity robot is a significant achievement in and of itself, and greatly improved upon the previous tensegrity robot used in our lab and used by Rieffel and Mouret in their work. Most significant was the move from tethered control to untethered, Bluetooth-based control. Previously, the tensegrity was attached to a breadboard circuit via magnet wire strung from the tensegrity’s motors to motor controllers on the breadboard, which were in turn controlled via USB from the supervisor machine. These tethers frequently tangled as testing progressed, requiring a human operator to consistently untangle the delicate wires. Additionally, because of the fragile nature of magnet wire, the process of untangling the tethers or occasionally just a particularly energetic gait would cause a tether to snap, necessitating a lengthy and arduous repair process. The tethers also exerted a slight force on the robot, particularly when tangled, which impacted the behavior of gaits. Moving to a wireless tensegrity has eliminated huge amounts of time spent performing maintenance on the robot, and has led to more reliable behavior data. It also, as noted above, is a necessary step towards producing a truly useful tensegrity robot. The second major improvement is the addition of instrumentation to the robot, which allows more complex control, including the possibility of closed-loop control based on sensed acceleration. The ability to not just sense but collect and store data also means that post-hoc analysis of the forces experienced by the robot is possible, which enables examination of the dynamic, mechanical properties of the robot.

2.2 Testing Setup

One side effect of the upgrade is that the robot itself became approximately twice as large, with the individual struts growing from 9.4cm in the previous version to 18.5cm in the current on. As a result, the previous 91x61cm testing table became too small for repeated testing without needing to center the robot after every test. In order to facilitate testing, we built a simple testing platform consisting of a wooden table with adjustable-length legs for leveling and an overhead camera used to track the movement of the tensegrity. The table features a flat testing area approximately 46x46in with a base of particleboard which fenced in by 1x6in boards to prevent falling. The platform is mounted on four approximately 1ft long 2x4in legs with adjustable-length feet at the bottom. The feet adjusted via a screw to allow the table to be leveled regardless of the unevenness of the floor. A 1080p HD Logitech webcam is hung above the platform on a custom mount which allows for multi-axis adjustments. The mount itself is attached to an aluminum extrusion beam which allows the camera to ba adjusted vertically and horizontally with ease. A 65-ft USB cable connects the webcam to a nearby desktop computer which acts as the supervisor. Testing is performed by placing the tensegrity in the center of the table and running a gait on the tensegrity for a brief period of time. The overhead camera feeds images into a tracking algorithm written in Python which tracks the motion of the tensegrity and report the total displacement at the end. The design and construction of the table was a joint effort between the author and Barhydt.

In addition to helping build the testing rig, the author developed a tracking algorithm which uses the OpenCV computer vision library to determine the location of the tensegrity in any given frame. The previous testing setup used colored cotton balls glued to the robot as tracking beacons, but this method was discarded in the new testing setup because it lacked reliability. The larger arena necessitated that the camera be higher above the board, and since the camera resolution was ultimately downsampled to 640x480 to improve computation time, the colored balls were harder for the camera to pick up and proved quite unreliable. This was compounded by the new location of the testing rig next to a hallway window, which resulted in varying lighting conditions on the testing arena.

Instead, a new algorithm was created which evolved from simple blob-tracking, which required a uniform, bright white background, to using image subtraction. Prior to tracking, the algorithm takes a picture of the arena completely empty, which it converts to grayscale, blurs twice, and then stores as a base image. Then, during tracking, it converts a given frame to black and white and then blurs the greyscale image twice, once with Gaussian blur and then by calculating the median value of the pixels around each pixel. The blurred image is subtracted from the base, tensegrity free image in order to identify pixels whose values have changed. The image difference is then thresholded twice. First, the value of each pixel is adjusted according to the piecewise function

$$\text{threshold-a}(v) = \begin{cases} 0 & v > 200 \\ v & v \leq 200 \end{cases} \tag{4}$$

which sets the value of each pixel to 0 (i.e., black) if its original value was greater than 200. This removes pixels whose difference was negative and thus resulted in extremely high difference values in the image. Then the resulting image is again thresholded, this time according the the piecewise function

$$\text{threshold-b}(v) = \begin{cases} 255 & v > 25 \\ 0 & v \leq 25 \end{cases} \tag{5}$$

which sets each pixel with a value above 25 to 255 and sets the rest to 0. This creates a purely black-and-white image which highlights all of the major differences still remaining after the first thresholding and discards the small differences probably due to image noise. Finally, the algorithm leverages OpenCV's built-in blob detection and tracking in order to find the largest area of white and calculate its center. The coordinates of the center are saved and made available for other code to use, and the original frame is altered to show all the contours OpenCV found as well as the perceived center of the tensegrity. The final image also displays a blue circle which indicates where the automated testing will pause to allow the tensegrity to be re-centered. The result of each stage of the tracking algorithm can be seen in Figure 3.

Actually running the gaits on the tensegrity is accomplished via communication with a desktop computer running the gait generation code. A Python API is used to interface with and send information to the tensegrity within the gait generation algorithms. This API is based on the Linux `gatttool` program which allows for command-line interaction with Bluetooth Low Energy devices such as the RFD77201. The API allows users to instantiate new `Strut` objects which have instance methods capable of sending or receiving raw data or performing certain predefined functions such as running a motor at a given speed. These methods work by constructing `subprocess` calls to `gatttool` based on user-specified parameters. The API cannot communicate with each strut concurrently, so any command sent to all struts must be sent in series to each individual strut. It is important to note that the effect of sending a particular datum to the RFD77201 depends on the program the RFD77201 is running, since different programs can interpret the same datum in different ways.

3 Experimental Design

In order to test how effective Bayesian optimization is for producing useful tensegrity gaits, we used four different gait generation methods: random search, Bayesian optimization using expected improvement as the acquisition function, and Bayesian optimization using upper confidence bound as the acquisition function with two different beta values: 0.5 and 1.5. We performed 20 gait optimizations with random search and expected improvement, and 10 each for the two different upper confidence bound functions. Each method and each optimization was composed of 60 ten second trials. Although this comes out to a minimum of ten minutes per optimization, optimizations often ran for closer to fifteen minutes due to delays caused by repositioning the tensegrity. Each trial was performed using the hardware and software described in the previous section to evaluate it, and without using input from the accelerometer or storing any data.

In order to perform the optimizations, we wrote a Python program which integrated the tracking code, the tensegrity API, and an out-of-the-box Bayesian optimization Python library called `PyGPGO`. The program first initializes the tracking code, then queries the optimizer for an initial set of motor speeds to try. It records the starting position of the tensegrity using the tracking code, then uses the tensegrity API to try the given set of frequencies. After ten seconds, the program confirms with each strut that the motors have stopped spinning and records the final resting position of the tensegrity. It calculates the distances between the starting and ending points in terms of pixels and returns this to the optimizer. If Bayesian optimization is being used, the optimizer incorporates the new datum into its internal probabilistic representation of the function between motor speeds and displacement, then generates a new set of frequencies to test. If random search is being used, the distance is saved but has no bearing to the next set of frequencies. Once a new set of frequencies is generated, the program repeats the process.

We initialized all of our Bayesian optimizers with a prior similar to that used by Rieffel and Mouret, rewarding having two or all three motors at full speed and penalizing having no motors moving. Specifically, out priors were

$$\begin{aligned} x_1 &= [0,0,0] & P(x_1) &= 0.0 \\ x_2 &= [255,255,255] & P(x_2) &= 50.0 \\ x_3 &= [255,255,100] & P(x_3) &= 75.0 \\ x_4 &= [255,100,255] & P(x_4) &= 75.0 \\ x_5 &= [100,255,255] & P(x_5) &= 75.0 \end{aligned}$$

This guided the algorithm to gaits with higher frequencies, and particularly to gaits where two frequencies were high and the third was only moderate, which hand-testing found to be particularly effective. The lowest possible value for the motors is 0, which stops the motor completely, and the highest is 255, which spins the motor as fast as it can clockwise. We were unable to use negative values because they cannot easily be send via the Bluetooth interface the RFDuino uses, and we chose to have a full range of speeds in one direction rather than having fewer speds but both directions. In the priors x_3, x_4, x_5, the lowest value is set to 100 because previous experience had shown that when two motors are spinning fast, the last motor ought to also be spinning, and spinning fast enough to induce vibrations. However, when all three are spinning at high speeds, the movement gets extremely chaotic and linear motion is rare.

User intervention is only required if the tensegrity gets too close to the edge. Between trials during the optimization process, the program checks whether the center of the tensegrity is farther than 115 pixels from the center of the frame. If it is, the program pauses and informs the user that the tensegrity its getting close to the edge and should be centered. It only resumes again once it detects that the tensegrity has been moved back inside the circle. On average, this occurs approximately every ten trials, but is highly dependent on the efficacy of the gaits tried: more effective gaits mean greater distances traveled and thus more time spent outside of the circle. The purpose of the safety zone is to ensure the tensegrity does not bump into a wall while trying a gait, since that would artificially lower the distance traveled by the gait.

At the beginning of each optimization, a new CSV file is created to collect the data produced by the optimization. After each trial, the program writes a new line of data to the file which includes information about the individual trial and the optimization as a whole. In regard to the trial itself, the frequencies tested in the trial, the distance traveled, the starting and ending locations, the trial number, and the best distance seen so far are saved. The program also records the optimizer and acquisition function being used, the minimum and maximum motor speeds, and the versions of the hardware and software being used.

4 Results

Prior to analyzing the data, the distances recorded were converted to number of tensegrity body lengths. Empirical testing was used to determine that the diameter of the tensegrity as perceived by the camera was, on average, 80 pixels. As such, distance values, which had been measured in pixels, were divided by 80 to determine the number of body lengths moved. In addition to providing a more grounded metric than pixel distance, which is entirely based on the type and placement of the camera, it also allowed for closer comparison with the work of Rieffel and Mouret, who used the same metric. As can be seen in Figure 4, Bayesian optimization using expected improvement as its acquisition function performs significantly better than either random search or Bayesian optimization using upper confidence bounds with beta values of 0.5 or 1.5. Whereas by trial 60 the median best distance discovered by random search is 0.71 body-lengths, Bayesian optimization using expected improvement yields a median best distance of 1.11 body-lengths. Bayesian optimization utilizing an upper confidence bound acquisition function is as ineffective as random search: using a beta value of 0.5, the median best distance was 0.69 body-lengths, less than random search. A beta value of 1.5 makes the situation is only slightly better, leading to a median best distance at trial 60 of merely 0.74 body lengths, barely higher random search.

The median and quartiles, as well as the inter-quartile range for each of the four approaches can be seen in Table 1 and are represented graphically in Figure 5. Bayesian optimization using expected improvement also surpasses random search and upper confidence bound based Bayesian optimization in terms of both the lower and upper quartiles. Additionally, using the Mann-Whitney U test, we determined that the differences in median best distance discovered by trial 60 between Bayesian optimization using expected improvement and each of its competitors were all statistically significant to at least the $p < 0.05$ level.

However, the spread of the expected improvement based Bayesian optimization is far greater than any of its competitors, with an inter-quartile range of 0.52 body-lengths compared to the 0.36 body-length spread of the next widest distribution, random search. In comparison, Bayesian optimization using the upper confidence bound with a beta of 0.5 had the narrowest spread, with only a 0.2 body-length difference between its quartile values. The wide spread of the expected improvement strategy becomes even clearer when considering Figure 5, which shows that the lowest final result of an expected improvement optimization was lower than any of its competitor's lower quartiles, while its most successful final result was more than 50% greater than that of its nearest competitor.

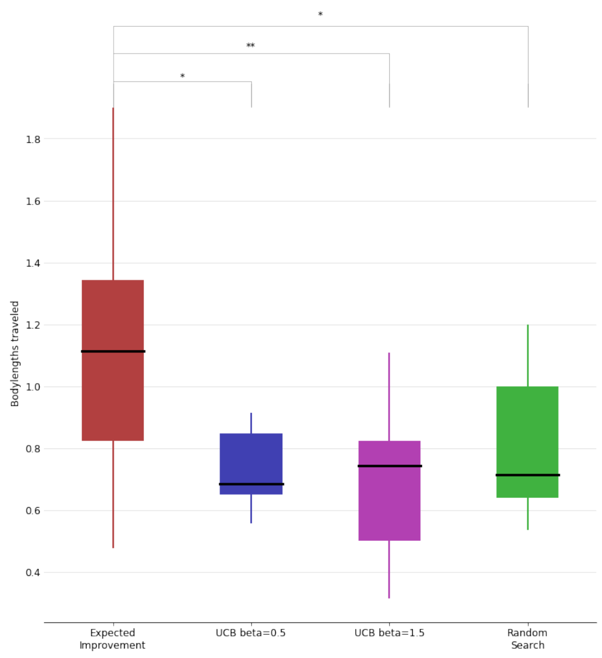


Figure 5: The statistical spread of the best distance traveled by the final trial for each of the four gait generation methods. The black line represents the median, the inter-quartile range is shown by the filled box, and the most significant non-outlying best distances are given by the whiskers. * = statistically significant with $p < 0.05$, ** = $p < 0.01$.

4.1 Discussion

The success of expected improvement based Bayesian optimization, particularly over random search, reinforces the conclusions of Rieffel and Mouret that Bayesian optimization is a promising tensegrity gait generation technique. However, our success was significantly more limited than that of Rieffel and Mouret, both in terms of the statistical significance of our results and the actual locomotion produced. In their work, Rieffel and Mouret show that Bayesian optimization using an upper confidence bound based

Optimization Method	Lower Quartile (25%)	Median	Upper Quartile (75%)	IQR	n
Bayesian (EI)	0.82	1.11	1.34	0.52	20
Bayesian (UCB $\beta = 0.5$)	0.65	0.69	0.85	0.2	10
Bayesian (UCB $\beta = 1.5$)	0.5	0.74	0.82	0.32	10
Random Search	0.64	0.71	1.0	0.36	20

Table 1: The distribution of best best distances in body-lengths traveled by trial 60 for each of the four gait generation methods explored. IQR is the inter-quartile range, giving some sense of the reliability of the algorithm.

acquisition function with beta at 0.2 is statistically significantly better than random search at the $p < 0.0001$ level, a powerful demonstration of the efficacy of Bayesian optimization. We were unable to replicate this level of confidence; rather, we found that Bayesian optimization using UCB as its acquisition function performs equivalently to or worse than random search. Only expected improvement based Bayesian optimization was shown to be significantly more effective than random search, and that only at the $p < 0.05$ level.

Additionally, the effectiveness of the gaits discovered via Bayesian optimization in Rieffel and Mouret [23] was far greater than the effectiveness of gaits found in our research. The median speed of gaits developed through the prior-based Bayesian optimization in Rieffel and Mouret's work is 11.5cm/s, which comes out to a relative speed of approximately 0.88 body-lengths per second. In contrast, the most effective gait developed during our testing traveled a total of 151.74 pixels during a ten second test, for a relative speed of 0.19 body-lengths per second, less than a quarter as fast as the gaits produced regularly by Rieffel and Mouret. This is additionally significant given that we ran twice as many trials per optimization, and ran each trial for more then three times as long. However, the tensegrity robot used by Rieffel and Mouret in their research was significantly different than the one used in ours. In particular, their robot was smaller and lighter due to the matiral it was made of and the lack of on-board electrnics. As such, it is not particularly concerning that our robot moved slower.

Nevertheless, our research does demonstrate that Bayesian optimization, at least when using expected improvement as its acquisition function, is particularly effective at generating tensegrity gaits. Although the gaits generated were significantly slower than those generated by Rieffel and Mouret, Bayesian optimization performed significantly better relative to the performances of the other methods in our research.

5 Troubles and Future Work

There are a number of potential causes which may have all contributed to the discrepancies between the data presented in Rieffel and Mouret [23] and the data presented here, all of which are related to the tensegrity system used in this research and all of which ought to be addressed in the future. The problems all relate to the robustness of the system and can be broken up into two groups: failures in hardware and failures in software.

5.1 Hardware Failures

The primary hardware failures afflict the tensegrity robot itself and arise from the intense and frequent vibration experience by the robot. Designing and building dynamic systems to cope with intense and consistent vibrations is notoriously difficult, and is a large part of the reason why other tensegrity actuation research focuses on spring length adjustment rather than vibrational actuation. In our system, the deleterious effect of the vibrations manifest through battery failures, circuit board fastener failures, and, most significantly, issues with the battery to board connections. Additionally, the testing table itself proved to be a consistent problem.

The least significant of the hardware issues were the failures of the nut and bolt holding the circuit board to the acrylic strut. Several times throughout testing the nut would be shaken loose from the bolt, allowing the battery clip and the bolt itself to separate from the tensegrity. This was, ultimately, a minor issue for the most part since each circuit board is attached with two fasteners, but it did exacerbate another issue, the battery problem, since the now-loose battery would be thrashed across the arena as the tensegrity moved. The battery failures occurred due to the design of the active struts pressing the lithium-ion battery on an active strut against the motor mount such that the wires extending from the batteries are forced to bend at sharp angles. This issue is compounded by the fact that the vibration causes the wires to rub against the sharp edge of the motor mount, degrading the quality of the wire and ultimately causing the wires to detach from the battery. We lost a total of five batteries during the testing process as a result of this issue. An additional hardware issue arose due to the difficulty in ensuring the testing arena was perfectly level, which caused some of the data initially gathered to be corrupt.

The most significant issue was somehow related to the connection between the batteries and the electronic components, although the exact nature of the problem has yet to be determined. The problem was that whenever a motor spun up to the high end of its speed range, there was a high likelihood the connection between the batteries and the circuit board would break, causing the RFDuino to power off. Most of the time the connection was reestablished nearly instantaneously, but occasionally it resulted in the connection needing to be fixed by hand before testing could continue. These failures had two effects of significance. First, they caused the strut experiencing the failure to stop spinning prematurely. effect occurred every time this sort of failure did because, on reboot, the RFDuino micro-controller had no way of returning to its previous rotational frequency. Instead, it would sit inert, waiting until the next trial for the supervisor computer to give it a frequency to test. Since these failures often occurred within the first couple seconds of a trial, this meant that the strut spent most of the trial inactive and thus the trial was not actually testing the set of three frequencies which the optimizer thought it was. Consider, for example, the trial frequencies

($m_1 = 230, m_2 = 180, m_3 = 90$). If the strut corresponding to m_1 failed quickly, the trial would really be testing the frequencies ($m_1 = 0, m_2 = 180, m_3 = 90$), but the optimizer would still take the results as genuine. Since gaits with even one non-rotating motor rarely produce any motion, the optimizer would quickly learn to avoid frequencies high enough to cause the failures, thus impacting its performance. Additionally, the connection would sometimes not be reestablished, necessitating hands-on troubleshooting. Since the tracking code detects objects via image subtract, it does not have a way to distinguish between a human hand or head and the tensegrity. Thus, hands-on troubleshooting always ran the risk of misleading the tracking software and causing an erroneous travel distance, corrupting the data.

5.2 Software Failures

The major software failure was the lack of robustness in the tracking software. As alluded to in the previous paragraph, the tracking software was unable to discern the difference between different kinds of objects and was not specially tuned to the tensegrity. As such, any sufficiently large change in the camera's view was considered a possible tensegrity, and if the area of the change was larger than the tensegrity, then it would be tracked, not the robot. This nearly always occurred in two different ways. The least common of the two and the easiest to remedy was human interference. In the initial phases of testing, lab regulars and visitors alike were curious about the robot and had a tendency to peer at the robot, bending their head over the table and obstructing the camera's view. Less frequently, someone would merely be standing to close to the table, accidentally casting to dark of a shadow on the table. An effective solution to both issues was carefully watching people near the testing table and regularly issuing warnings not to approach to closely.

The second and much more frustrating issue stemmed from the fact that, at a certain point in the evening, the hallway lights become motion-activated and turn off after a certain period of inactivity in the hallway. Due to the location of the testing table next to a large window and between, rather than under, two lights in the lab, much of the lighting on the table was determined by the lighting in the hallway. Since a base image was taken only once, at the beginning of the optimization, a significant change in hallway lighting would often result in a significant change in the camera's view that the tracking algorithm would pick up and track instead of the tensegrity. Thus, it was necessary to constantly ensure the hallway lights were on when testing in the evening or at night, and the only means of doing this was activating the motion sensors controlling the light. Two other factors made the issue even worse: the inability to save and load optimizations and the fact that classes and daily life meant much testing was done at night. The inability to save, load, or restart an optimization was particularly troublesome because it meant that if the hallway lights shut off at an inopportune moment, an erroneous travel distance would be recorded and the entire optimization would become corrupted and there was no way of recovering previous progress.

Several means of fixing this problem exist and ought to be examined and, if feasible, implemented in the future. First, some sort of cloth or paper could be used to obscure the window in question when testing, which would drastically reduce the influence the hallway lighting has. This would be considerably easier than relocating the testing rig, whose size makes it both unwieldy to move and an inconvenient obstacle most places in the lab. Another option is to mount a set of powerful light sources above the testing arena to provide uniform lighting bright enough to reduce the influence of the hallway lighting. Finally, the tracking algorithm could be modified to better handle changed in lighting, or scrapped altogether for a more robust method. Another, minor problem with the tracking algorithm as implemented is that it crashes if it cannot find any object to track in the frame. Although the solution to this is relatively trivial, it rarely occurred since we were aware of the issue and careful not to let it happen.

5.3 Future Research

Aside from fixing the flaws exposed during our testing, this research presents several potential threads of research to follow. Continued work with both Bayesian optimization could prove fruitful, particularly after the issues highlighted previous are addressed, because the results of this research project, while promising, are hardly conclusive or robust. Bayesian optimization involves hyper-parameters such as the surrogate process used, the acquisition function used, and the number and length of trials, in addition to any hyper-parameters the acquisition function introduces such as the beta value used by UCB, which ought to be explored and tuned. Other optimization algorithms such as Covariance Matrix Adaptation Evolution Strategy ought to also be explored as potentially effective and efficient gait generators, and other machine learning techniques, particularly neural networks, could also proof useful.

Neural networks bring to the table another research direction discussed earlier in Section 1.1: the use of central pattern generators to produce dynamic, rather than static, gaits. Spiking neural networks potentially offer a responsive and dynamic closed-loop control system for tensegrities, and research in this area is already on-going. Finally, it could be interesting to combine the use of CPGs with the idea of morphological computation discussed by, among others, Rieffel et al. [21]. Utilizing the dynamic properties of a tensegrity in order to off-load control calculations could present a particularly effective control strategy.

6 Conclusion

Although not conclusive, the research described here has paved the way for increasingly interesting and varied investigations into tensegrity robots. First, we have developed a new, modular tensegrity robot equipped with data collection and wireless communications capabilities, a significant advance from our previous tensegrity robot. Supporting this development, we've created an entirely new testing system, including both a physical testing table and a suite of software enabling nearly any kind of testing needed. We utilized these new capabilities to assess and affirm previous research on tensegrity gait generation undertaken by Rieffel and Mouret [23] that Bayesian optimization is a promising means of generating effective gaits on tensegrity robots. In particular, we have demonstrated with some confidence that Bayesian optimization is significantly more capable of generating useful gaits than random search, and specifically that expected improvement ought to be used as the acquisition function for the optimization. Equally important, through this research we have discovered a number of flaws and failures in our new setup which must be addressed, and in doing so paved the way for more effective research in the future.

References

[1] Adrian Agogino, Vytas SunSpiral, and David Atkinson. "Super Ball Bot-structures for planetary landing and exploration". In: *NASA Innovative Advanced Concepts (NIAC) Program, Final Report* (2013).

[2] Thomas K Bliss. "Central pattern generator control of a tensegrity based swimmer". PhD thesis. University of Virginia, 2011.

[3] Thomas K Bliss, Tetsuya Iwasaki, and Hilary Bart-Smith. "Central pattern generator control of a tensegrity swimmer". In: *IEEE/ASME Transactions on Mechatronics* 18.2 (2013), pp. 586–597.

[4] Thomas K Bliss, Tetsuya Iwasaki, and Hilary Bart-Smith. "Resonance entrainment of tensegrity structures via CPG control". In: *Automatica* 48.11 (2012), pp. 2791–2800.

[5] Valter Böhm and Klaus Zimmermann. "Vibration-driven mobile robots based on single actuated tensegrity structures". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 5475–5480.

[6] Ken Caluwaerts et al. "Locomotion without a brain: physical reservoir computing in tensegrity structures". In: *Artificial life* 19.1 (2013), pp. 35–66. URL: http://www.mitpressjournals.org/doi/abs/10.1162/ARTL_a_00080.

[7] Samanwoy Ghosh-Dastidar and Hojjat Adeli. "Spiking neural networks". In: *International journal of neural systems* 19.04 (2009), pp. 295–308. URL: http://www.worldscientific.com/doi/abs/10.1142/S0129065709002002.

[8] Helmut Hauser et al. "The role of feedback in morphological computation with compliant bodies". In: *Biological Cybernetics* 106 (2012), pp. 595–613.

[9] Helmut Hauser et al. "Towards a theoretical foundation for morphological computation with compliant bodies". In: *Biological Cybernetics* 105 (2011), pp. 355–370.

[10] Auke Jan Ijspeert. "Central pattern generators for locomotion control in animals and robots: a review". In: *Neural networks* 21.4 (2008), pp. 642–653.

[11] Atil Iscen et al. "Controlling tensegrity robots through evolution". In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM. 2013, pp. 1293–1300. URL: https://dl.acm.org/citation.cfm?id=2463525.

[12] Atil Iscen et al. "Flop and roll: Learning robust goal-directed locomotion for a tensegrity robot". In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE. 2014, pp. 2236–2243. URL: http://ieeexplore.ieee.org/abstract/document/6942864/.

[13] Mark Khazanov, Jules Jocque, and John Rieffel. "Evolution of locomotion on a physical tensegrity robot". In: *ALIFE*. Vol. 14. 2014, pp. 232–238.

[14] Mark Khazanov et al. "Exploiting Dynamical Complexity in a Physical Tensegrity Robot to Achieve Locomotion." In: *ECAL 2013*. Union College. 2013, pp. 965–972. URL: `http://cs.union.edu/~rieffelj/papers/rieffel-khazanov-tensegrity.pdf`.

[15] Guang Lei Liu et al. "Central pattern generators based on Matsuoka oscillators for the locomotion of biped robots". In: *Artificial Life and Robotics* 12 (2008), pp. 264–269.

[16] Brian T Mirletz et al. "Design and control of modular spine-like tensegrity structures". In: *Proceedings of The 6th World Conference of the International Association for Structural Control and Monitoring* (2014).

[17] Brian T Mirletz et al. "Goal-directed cpg-based control for tensegrity spines with many degrees of freedom traversing irregular terrain". In: *Soft Robotics* 2.4 (2015), pp. 165–176. URL: `http://online.liebertpub.com/doi/abs/10.1089/soro.2015.0012`.

[18] Chandana Paul. "Investigation of Morphology and Control in Biped Locomotion". PhD thesis. University of Zurich, 2004.

[19] Chandana Paul. "Morphological computation: A basis for the analysis of morphology and control requirements". In: *Robotics and Autonomous Systems* 54.8 (2006), pp. 619–630.

[20] Chandana Paul, Francisco J Valero-Cuevas, and Hod Lipson. "Design and control of tensegrity robots for locomotion". In: *IEEE Transactions on Robotics* 22.5 (2006), pp. 944–957. URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1705585`.

[21] John A Rieffel, Francisco J Valero-Cuevas, and Hod Lipson. "Morphological communication: exploiting coupled dynamics in a complex mechanical structure to achieve locomotion". In: *Journal of the royal society interface* 7.45 (2010), pp. 613–621. URL: `http://rsif.royalsocietypublishing.org/content/7/45/613.short`.

[22] John A Rieffel et al. "Locomotion of a tensegrity robot via dynamically coupled modules". In: *Proceedings of the International Conference on Morphological Computation*. 2007. URL: `http://cs.union.edu/~rieffelj/papers/MC07_Rieffel.pdf`.

[23] John Rieffel and Jean-Baptiste Mouret. "Adaptive and Resilient Soft Tensegrity Robots". In: *Soft robotics* (2018), pp. 318–329.

[24] Brian R Tietz et al. “Tetraspine: Robust terrain handling on a tensegrity robot using central pattern generators”. In: *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*. IEEE. 2013, pp. 261–267. URL: http://ieeexplore.ieee.org/abstract/document/6584102/.

[25] Michael T Turvey and Sérgio T Fonseca. “The medium of haptic perception: a tensegrity hypothesis”. In: *Journal of motor behavior* 46.3 (2014), pp. 143–187. URL: http://www.tandfonline.com/doi/abs/10.1080/00222895.2013.798252.

[26] Marvin Zhang et al. “Deep reinforcement learning for tensegrity robot locomotion”. In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 634–641. URL: http://ieeexplore.ieee.org/abstract/document/7989079/.

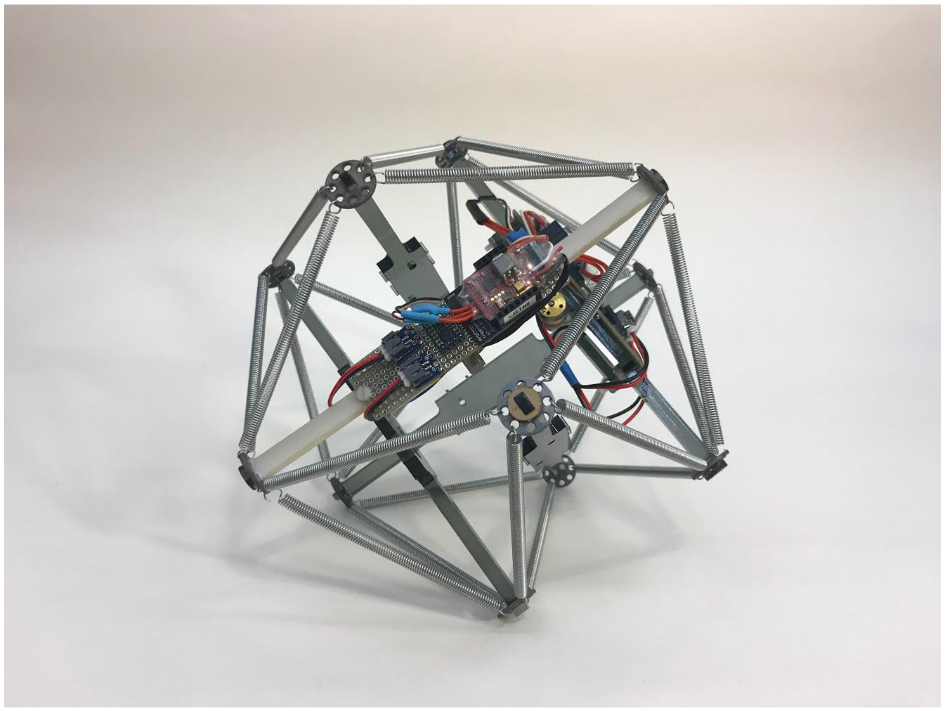


Figure 1: A picture of our current tensegrity robot: VVValtr (VVireles Vibrationally Active Limbless Tensegrity Robot.

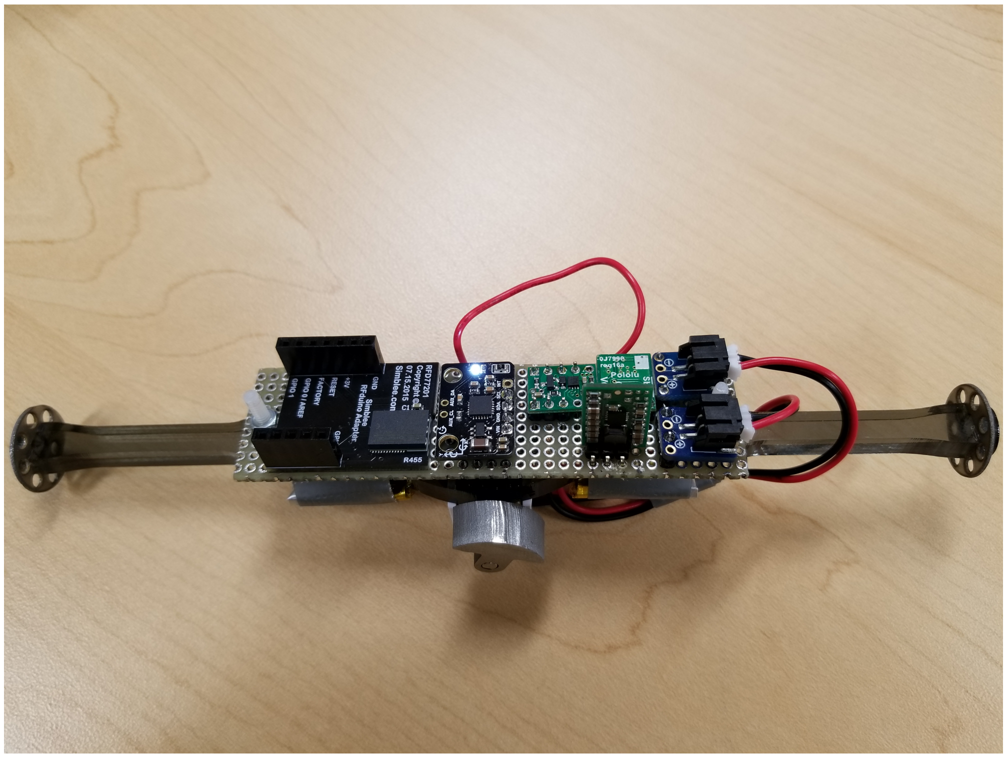
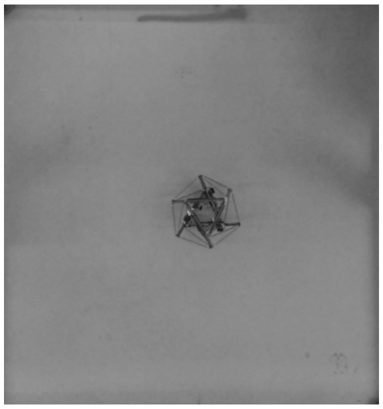
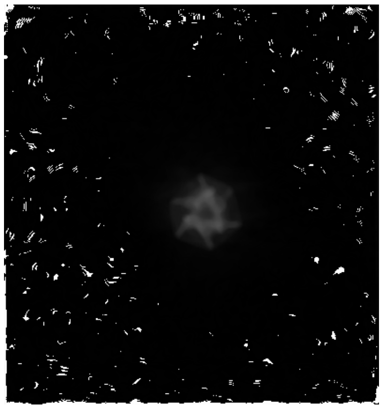
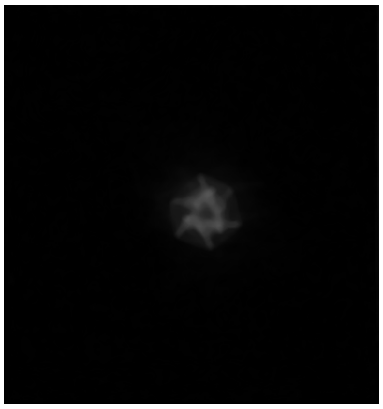


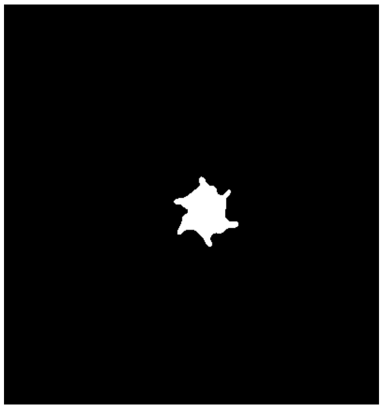
Figure 2: A picture of one of the struts which make up VVValtr.

(a)


(b)


(c)


(d)


(e)

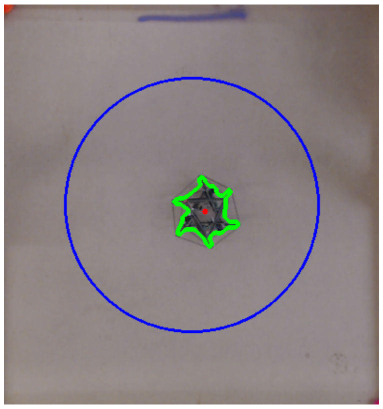

(f)

Figure 3: Visualizations of the tensegrity tracking algorithm. Figure 3a shows the base image taken prior to tracking. In Figure 3b, the grayscale frame being analyzed is shown. Figure 3c illustrates the resulting difference once the current frame is subtracted from Figure 3a. Figure 3d and Figure 3e each show the result of the successive thresholds, with Figure 3d removing extraneous high-value artifacts and Figure 3e isolating and highlighting the tensegrity. Finally, Figure 3f is the image shown to the user.

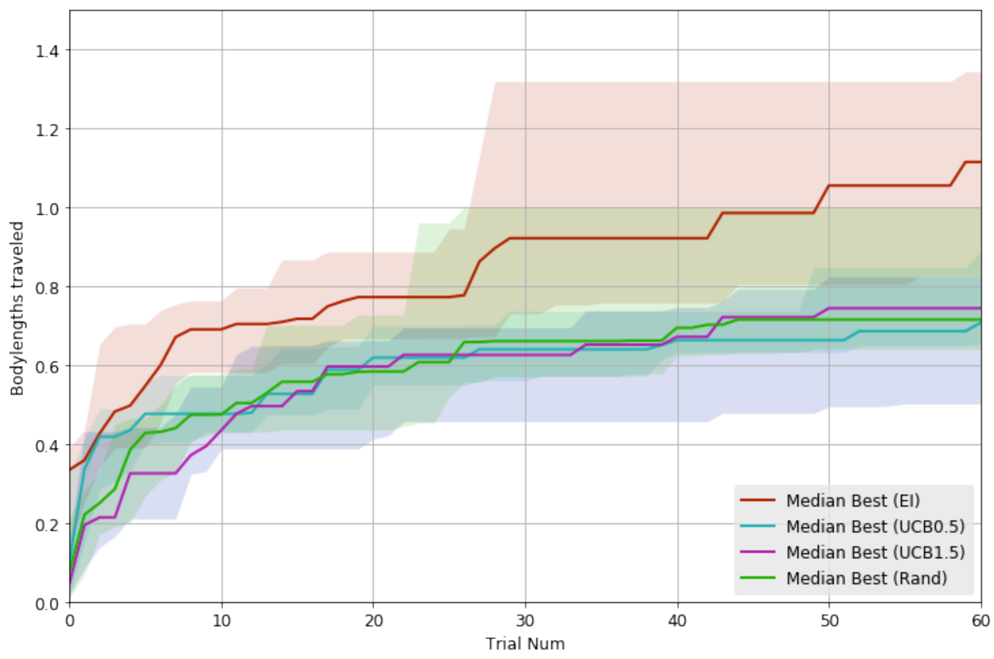


Figure 4: This graph illustrates the improvement in gait generation over time for each of the four different gait generation methods used. The dark lines follow the median value of all of the best distances achieved by each run of each gait generation function at by each trial. Thus, for example, by trial 10, the expected improvement method had a median best distance traveled of approximately 0.7 body-lengths. The graph demonstrates that the expected improvement method performed better than the other three methods.

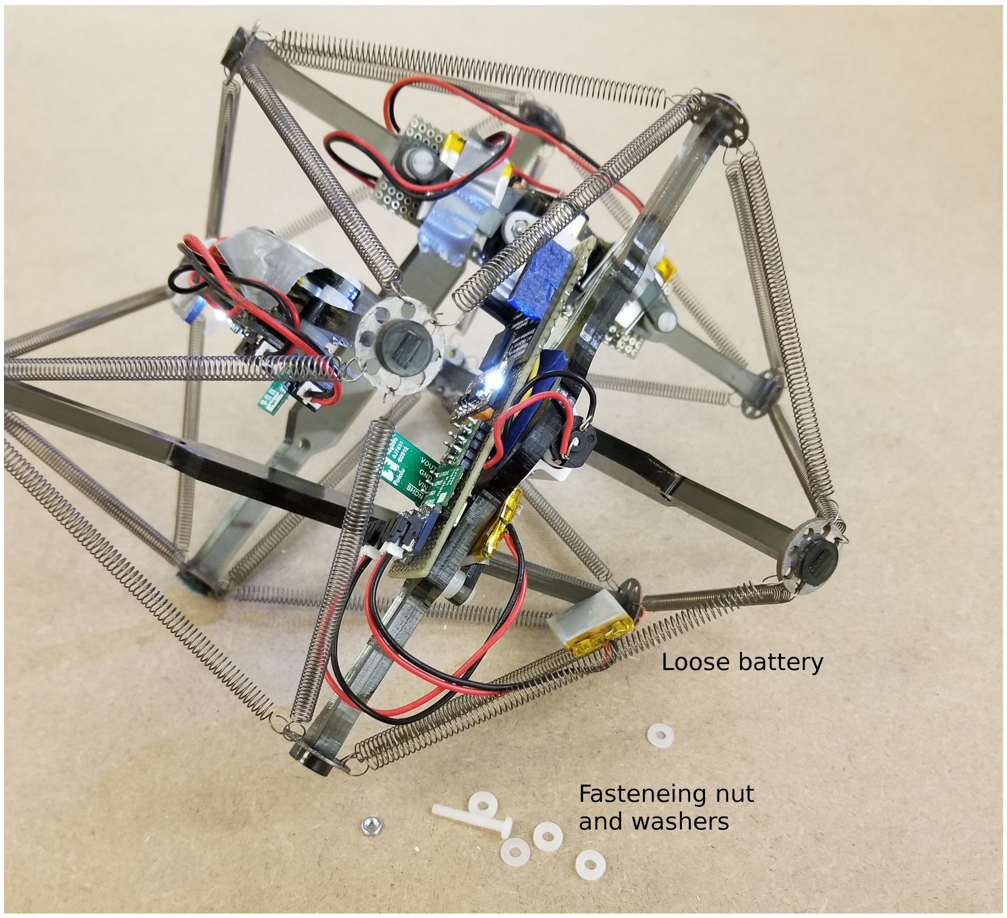


Figure 6: The result of a circuit board fastener failing. The fastening nut and it washers are labelled, as is the loose battery resulting from the failure. Although unlabeled, the circuit board can also be seen to be somewhat loose.

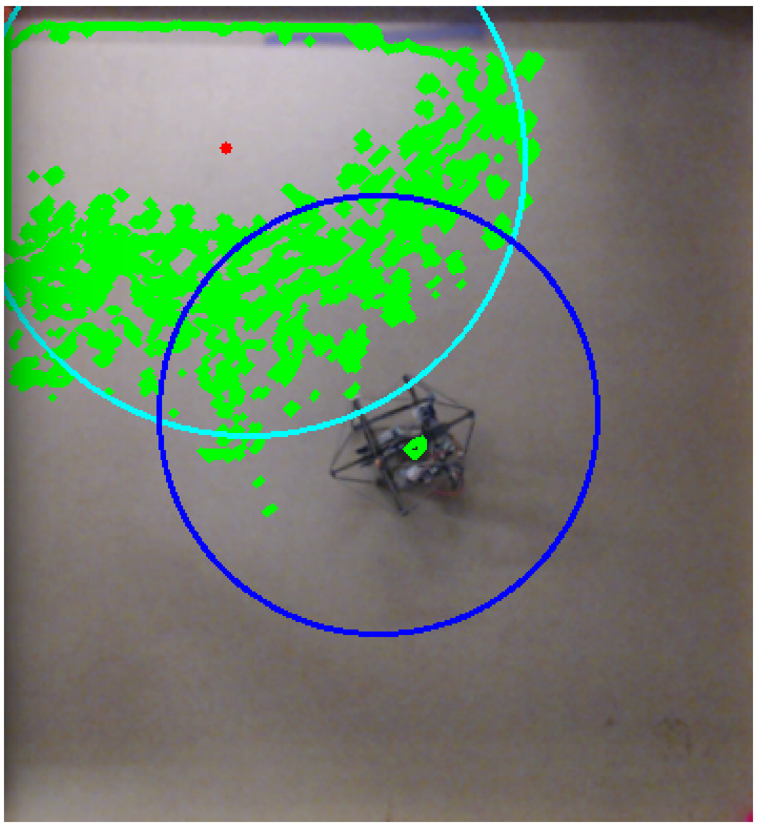


Figure 7: An example of how a change in lighting can mess up the tracking software. In this picture, the hallway light had previously been off but turned on when someone was walking down the hall, causing a bright patch which threw off the tracking code.